Probing a Collection to Discover Its Language Model

Aiqun Du and Jamie Callan
Computer Science Department
University of Massachusetts
Amherst, MA 01003

{adu,callan}@cs.umass.edu

Abstract

Most solutions to distributed IR rely on access to a language model for each text collection, but it has been unclear how the model can be obtained reliably in real-world distributed environments. This paper proposes a solution based upon probing the collection, and demonstrates its effectiveness on four databases.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE
1998

2. REPORT TYPE

3. DATES COVERED
00-00-1998 to 00-00-1998

4. TITLE AND SUBTITLE
Probing a Collection to Discover Its Language Model

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Massachusetts Amherst,Department of Computer Science,140 Governors Drive,Amherst,MA,01003

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**20**	

1 Introduction

Distributed information retrieval systems assume more and more importance as information resources proliferate on internets and intranets. One of the new issues in distributed IR is how to do collection selection for online collections for which we don't have access to the full text, i.e., the documents in the collection are hidden behind a query interface. For example, a newspaper website might have a database of archived articles and a user can retrieve articles by querying the database, but there is no way to directly get all of the articles. Previous research on collection selection has been based on the assumption that the full contents and exact statistics of the collections are readily available. However this is not applicable to real Web environment with hundreds or thousands of collections managed by many different independent information services. To do content-based collection selection without cooperation in such environment, before deciding which collections to search, the first problem is how to find out what a collection contains, i.e., to automatically create a representation that well represents the content of the collection.

In addition to collection selection in distributed searching, this problem also has significance in many other applications that need to know the content of a database which is not possible other than through query. For example, a system that assists children to locate information in networked environments needs to identify which Web databases are useful or appropriate for children of various ages. It can't simply ask the databases themselves, so it needs a method to find out what each database contains. Another example is browsing, i.e., to show a list of the frequent words that occur in a database, so as to understand what's in the database.

We propose to solve this problem by probing the database, that is, to send "probe" queries to the database, to get a representative sample of the documements in the collection. The retrieved documents are examined to build a learned dictionary that represents the collection's language model. The constraint in building a collection model is that we know nothing about the collection except what is found by probing. We have experimented with several probing methods on a number of test databases, and found that the probing approach can build a learned dictionary that accurately represents the content of the original collection.

The rest of this paper is organized as follows: Section 2 discusses related work in collection selection for distributed IR. Section 3 presents our hypothesis in collection probing. Section 4 introduces the experimental method and evaluation measures. Section 5 describes the experiments performed and gives a detailed analysis of the results obtained. In the final section, we draw conclusions and identify possible future work.

2 Related Work

Collection selection is the first step of distributed searching. For environments where there are many sites, such as wide-area networks or the Internet, one approach is content-based collection ranking and selection. It consists of ranking collections for relevance to a query, and then selecting the best subset from a ranked list.

A widely used technique for selecting among many collections is to search a centralized collection selection index. Typically, the collection selection index consists of a set of lightweight representations for the collections, each of which consists of the collection's vocabulary and its word frequencies. This method of describing the contents of distributed collections has several advantages:

- It is cheap since the centralized index is relatively small; its moderate storage requirement (less than 0.4% the size of the original collection [2]) makes it easy to maintain.
- It is better than only indexing a small portion of each document such as titles, since the latter approach loses important information about collection content.
- Automatic creation of indexes rather than manual creation of summaries for individual collections guarantees consistancy.
- Creation of indexes is completely automatic without any manual effort, hence scales well to widely distributed and dynamic collections.

This technique was used in GLOSS [6][7], which was later extended to the vector space model in gGLOSS [5], which was further extended to the hierarchical GLOSS – hGLOSS. GLOSS estimates the number of potentially relevant documents in collection C for a boolean AND query Q using two kinds of collection statistics: the number of documents in C containing each term in Q, and the total number of documents in C. The GLOSS approach is easily applied to large numbers of dynamic collections in realistic distributed environment, because it stores only term frequency information (e.g., a term's document frequency) about each collection. A preliminary evaluation of gGLOSS reported in [3] examined the effectiveness of the gGLOSS approach to collection selection, and found that its collection ranking is reasonably accurate under certain conditions.

Callan et al. [2] also used the words that occur in a collection and their frequencies to describe the content of the collection for the collection selection index. Collection ranking is based upon the statistics including document frequency (the number of documents containing the term) and collection frequency (the number of collections containing the term). Their work differs from GLOSS in two major aspects. First, Callan's system is based on the inference net, a probabilistic retrieval model, whereas GLOSS uses a boolean retrieval model (gGLOSS is based on the vector space model). Second, in Callan's work, one retrieval algorithm is used for ranking both collections and documents, based on the argument that ranking collections is analogous to ranking documents; in contrast, GLOSS uses different algorithms for these two kinds of retrieval.

Both approaches require knowing the words and word frequencies in a collection. How this information is obtained is an open problem. One approach is for each collection to supply this information about itself, as proposed by the STARTS protocal [4]. However, this approach assumes that each collection is capable of and willing to, supply this information. It has so far been unclear how to integrate older legacy systems, systems that simply refuse to cooperate, and systems that misrepresent themselves (e.g., "spamming" on the Internet).

3 Hypothesis

Most large collections of text documents have similar statistical characteristics. Zipf's law [12] shows that words are not distributed evenly; a few words occur very often, whereas most words are infrequent. Luhn [8] proposed that frequency of word occurrence furnishes a useful measurement of word significance. Common words are generally stopwords. Rare words that occur only once or twice are mostly strange names, misspellings, etc. Natually these two kinds of words are non-significant words. The set of words with frequency lies in between them are significant words in that they contribute significantly to the content of the document. We call those words content words and hope to find most of them in order to estimate the content of the collection. Since content words typically have medium to high frequency, the probability of finding most of them by probing is high.

The aim of probing a database is to build a model that reasonably accurately represents the content of the database. The model is in the form of a learned dictionary containing a list of words that occur in the documents found by probing. The set of words in the learned dictionary is only a subset of that in the real dictionary, since one can only get a subset of the collection through queries. Although it may be computationally possible to get all of the words by using many probes, it is infeasible in practice, because the time it takes can be undeterministically long, and the probing termination condition cannot be decided since the actual size of the collection is unknown. More importantly, finding all or even most of the vocabulary is not necessary, since Zipf's law tells us that each collection, no matter what its size, has a fairly small "working vocabulary".

Probing is like taking a sample of a collection. As we do more probes, our sample becomes larger, and hence more representative of the collection. We can calculate statistics such as the ranks of words based on the sample, to estimate their true ranks in the original collection. Since the size of the collection is constant, as we take a larger sample, the estimates become more accurate, i.e., the ranks of the words converge to be close to their true values. If the sample could be chosen perfectly randomly, the estimates would be relatively accurate. Therefore an accurate model representative of the collection content can be built by probing.

After the first round of probes, we have seen a small collection of documents. We can calculate the ranks of the words we've seen. Due to sampling error, our calculations are just estimates with confidence intervals around them. After the second round of probes, we have seen a larger collection of documents. Again we calculate the ranks of the words we've seen. It is expected that our confidence intervals should shrink. When there is substantial variation in estimates from one probe cycle to the next, the confidence intervals are large; when the variation is small, it is probable that the estimates have converged to be close to the true values.

4 Experimental Method

4.1 Probing Method

A learned dictionary consists of a list of words that occur in the sample documents and their frequency of occurrences. Initially the learned dictionary is empty representing that we know nothing about the database. The dictionary grows as new words are found, and old words' frequency statistics are updated when they are encountered again. As we get a larger sample through probing and update the dictionary accordingly, our estimates of the ranks of terms in the original collection using the learned dictionary becomes more accurate . We considered three ways of calculating the ranks of terms in the dictionary using three types of frequency metrics: collection term frequency *ctf* (number of times the term occur in the sample documents), document frequency *df* (number of documents in the sample containing the term), and average term frequency *atf* (collection term frequency divided by document frequency).

The iterative probing process consists of following steps:

1. Generate a set of *M* initial probe words.
2. Run *M* one-word probe queries. Get the top *N* documents retrieved by each query.
3. Parse the documents to make a list of the words in the documents (excluding document markups, e.g., SGML or HTML tags), and how often each occurs. Update the dictionary.
4. Decide whether to stop probing; if not,
5. Generate a new set of *M* probe words. Go to Step 2.

Our primary concern is the methods of generating initial probe words in Step 1 and new probe words in Step 5. Initial probe words can be generated by choosing words from a well-known large collection, for example, the 3GB TREC corpus (volume 1, 2, 3). New probe words can be chosen from two kinds of sources, either from the learned dictionary, or from a general collection such as the TREC corpus. We experimented with different methods of choosing *M* probe words from a dictionary, such as choosing the *M* most frequent words or just a random selection. There are further variations in deciding the most frequent words using different ranking criteria, i.e., ranking by *ctf*, *df*, or *atf*. As a convention, we use *Ctf*, *Df*, and *Atf* to represent the methods of choosing new probe words corresponding to the three ranking criteria. In addition, we define *Random* and *Random2* to represent the methods of randomly selecting words from the learned dictionary and from the TREC corpus, respectively. The three methods of *Ctf*, *Df*, and *Atf* have a sort of learning property in that, later probe queries come from the most frequent words in the sample documents that have been obtained in earlier probes. They use the previous probing results as a kind of feedback in generating new probe queries, whereas the two random methods don't have such characteristic. We are interested in investigating how these five different methods work and how the random approach compares with the most-frequent-words approach. Other issues include how many documents to retrieve per query, and when to stop probing.

On the Internet, we won't know what the stopword list and stemming algorithm are for the database we are probing. So we don't use a stopword list or a stemming algorithm when parsing the documents that a database returns. Only at the end of probing, a stopword list is used to remove words in the learned dictionary that tell little about content. In doing evaluation, when the stemming algorithm of the test database is known, the same stemming algorithm is used to post-process the final learned dictionary, before comparing the learned dictionary to the real dictionary.

4.2 Evaluation Measure

The effectiveness of probing was tested first using the INQUERY retrieval system and three different TREC collections, and later using a real Web database service. INQUERY is based on the Bayesian inference net model and is described in detail in [10][9][1]. For the purpose of evaluation, we used INQUERY indexing system to build databases from the TREC collections, so that the known stopword list and stemming algorithm can be used for both the learned dictionary and the real dictionary in order to make a comparison between them. We propose the following measures to evaluate the effectiveness of probing (assuming the real collection statistics are known).

- percentage of terms found by probing

  It is used to show the rate of finding new terms, or the pattern of growth of the learned dictionary size with respect to the number of documents examined.

- *ctf* proportion

  ctf proportion refers to the proportion of the term occurrences of those terms found by probing in the total term occurrences of the collection. For example, a *ctf* proportion of 80% means that the learned dictionary contains the words that account for 80% of the content (word occurrencies) of the original collection.

  For a real dictionary D and a learned dictionary D', *ctf* proportion is calculated as:

  $$\frac{\sum_{i \in D'} O\_ctf_i}{\sum_{i \in D} O\_ctf_i}$$

  where O_ctf_i is the number of times term i occurs in the original collection.

  ctf proportion shows the frequency characteristics of the words found by probing. A high *ctf* proportion means most of the words found are frequent words. According to Zipf's law, frequent terms constitute a large proportion of the total term frequency; for example, top 50 words can account for 50% of the total word occurrences. So this metric can be used to tell whether (or roughly when) most of the frequent words have been found by probing.

- mean squared error on rank estimates

  The mean squared error metric was used to compare the estimated term rank to the true rank. It is a simple and effective way of measuring how closely the ranks of the terms in the learned dictionary match their ranks in the real dictionary.

  Ranking is based upon a term's collection term frequency (*ctf*). The most frequent term is ranked 1, and the least frequent is ranked R, where R is the number of distinct (unique) ranks. If two terms have the same *ctf*, they get the same rank. The scaled rank for a term is $rank/R$, so the top ranked term has a scaled rank of $1/R$, and the terms that occur just once have a scaled rank of 1. The scaled rank has two advantages: it is less dependent on database size and, it applies equally to frequent and rare terms.

  The mean squared error on rank estimates for a learned dictionary D' is calculated as:

  $$\frac{1}{n} \sum_{i \in D'} (Est\_rank_i - Real\_rank_i)^2$$

  where n is the number of terms in the learned dictionary, Est_rank_i is term i's estimated rank in learned dictionary, and $Real\_rank_i$ is term i's real rank in real dictionary.

- mean squared error on *idf* estimates

  The mean squared error metric was used to compare the estimated *idf* (inverse document frequency) value to the true *idf* value.

  Given a collection of documents, a term's *idf* is defined as:

  $$log(N/df)$$

  where N is the total number of documents in the collection, and df is the number of documents containing the term. An unscaled *idf* is used so that the mean squared error is relatively independent of the number of documents in the collection.

  The mean squared error on *idf* estimates for a learned dictionary D' is calculated as:

  $$\frac{1}{n} \sum_{i \in D'} (Est\_idf_i - Real\_idf_i)^2$$

  where n is the number of terms in the learned dictionary, Est_idf_i is term i's estimated *idf* based on the collection of sample documents, and $Real\_idf_i$ is term i's real *idf* based on the original collection.

5 Experiments

5.1 Test Collections and Experimental Setup

The effectiveness of probing was evaluated using the INQUERY retrieval system and three TREC collections that differ significantly in size and nature:

- CACM: a very small database containing abstracts of scientific articles on computing;
- WSJ88 (the 1988 Wall Street Journel): a medium-sized American newspaper database; and
- TREC-123 (the TREC corpus, volume 1, 2, and 3): a large and heterogeneous database containing 17 subcollections from different sources and/or period of time.

Table 1 lists some statistics about the three databases.

Table 1: The three test databases

Collection	*Size*	*#Docs*	*#Unique Terms*	*Total Term Occurrences*
CACM	2MB	3,204	6,468	117,473
WSJ88	104MB	39,904	122,807	9,723,528
TREC-123	3.2GB	1,078,166	1,134,099	274,198,901

The probing system runs 100 (10 for CACM) one-word queries per probe cycle, and examines the top 10 documents retrieved by each query. The set of initial probe words are generated using the dictionary of TREC-123 after filtering out numbers and terms whose occurrence is less than 6 (to reduce query failure). New probe words are chosen without considering very short words such as 1-or-2-character words since they tend to be stopwords. Numbers are discarded in parsing the documents retrieved, and are not included in the learned dictionary. This is because numbers occur often but are useless for estimating collection content. The probing system runs till at least 3000 database documents are seen (WSJ88 and TREC-123), or till 50% of the documents are seen (CACM). The number of probe queries used, the number/percentage of documents examined, and the number/percentage of words found (excluding stopwords), in testing the three databases using the *Df* method are given in Table 2.

Table 2: Probing experiments statistics

Collection	*# Queries*	*#(%) Docs Examined*	*#(%) Terms Found*
CACM	411	1,602 (50%)	3,952 (68%)
WSJ88	564	3,000 (7.5%)	19,600 (25.1%)
TREC-123	518	3,000 (0.3%)	47,320 (6.4%)

Each of the five methods of generating new probe words was tested and an analysis of their effectiveness based on the results follows. We have also experimented with different methods of choosing initial probe words, including choosing the most frequent words based on *ctf*, *df*, or *atf*, or a random selection. We found that there is no obvious difference among the evaluation results of these methods, which implies that the variation in the way of generating the initial probe words does not change the effectiveness of probing. So generally we choose the initial probe words based on *ctf*.

Evaluation was automatically performed at the end of each probe cycle using the measures described above. Figures 1 to 4 show the evaluation results for probing WSJ88. Figures 5 to 8 are for CACM, and Figures 9 to 12 are for TREC-123. The stopword list used was the INQUERY stopword list which consists of 418 common words that are generally only of syntactic utility.

5.2 Analysis of Results

Figure 1 shows that the rate of finding new words keeps high for the first several probe cycles, and gradually lowers down thereafter; new words occur less frequently as the sample grows. Up to 50% of the terms are found when only 10% of database documents are seen. This demonstrates that we can find a considerable proportion of the terms in the database by examining a relatively small number of documents.

In Figure 2, The *ctf* proportion quickly grows to be higher than 90% when roughly 3000 documents are seen, and then the growth rate drops sharply. This means the words found by the first several probes are mostly frequent words and the words found by subsequent probes tend to be rare words, since by Zipf's law, a small number of frequent words constitute a large proportion of the total term occurrences. This result confirms that most of the frequent words can be found by probing.

In Figure 3, mean squared rank error quickly drops down to below 0.02 when 3000 documents are seen, and converges to be close to the optimal value of 0. The very low error demonstrates that the probing techniques can accurately estimate term rank. The very low mean squared *idf* error shown in Figure 4 further indicates that probing can also accurately estimate term frequency statistics such as *idf*. Although doing more probes will possibly find more terms and thereby further reduces estimates error, it is likely to yield diminishing benifits. As shown in Figures 1 to 4, only minor improvements are achieved at a significantly higher cost after a certain number of documents are seen.

Although the five methods all do a reasonably good job (low error in rank estimates, can find most of the frequent words, etc.), it seems that they do not perform eaqually well. By comparison, *Random2* gives the best curves in Figures 1, 2, and 3; when the same number of documents are examined, it finds more terms, attains a higher *ctf* proportion, and has a lower mean squared rank error than other methods. The curves of *Atf* and *Random* appear to be relatively closer to each other, as are the curves of *Ctf* and *Df*.

Not so surprisingly, the random methods turn out to be better than the non-random ones. This may be due to the fact that, they use probe words that are selected randomly rather than from the most frequent. Random probe words have a higher probability of hitting documents scattered throughout the collection, making it more likely to sample the database in a random fasion. This confirms that it is better to sample as randomly as possible.

When the same number of documents are seen, *Random2* is able to find more important words. However, in order to retrieve a same number of documents, *Random2* needs to run more probe queries than other methods, since it has a higher query failure. This is due to the probe words chosen by random from TREC-123 contain a fair number of rare words, such as special names, which occur very frequently only in some documents in TREC-123. In terms of the number of queries used, *Random2* tends to be slower to converge to the optimum model; on the other hand, in terms of the number of documents examined, *Random2* converges more quickly than other methods.

Although the set of words found by different probing methods are not identical, there exists considerable overlap among them. Furthermore, if we rank the words using the same frequency metric such as *ctf*, the resulting lists of different methods are rather similar to each other, especially the top 50 words. The top portion of the list plays a more important role in browsing since statistically the most frequent words (except stopwords) are more representative of the content. Probing results demonstrate that the frequent words found can help us understand what the database is about. For example, the top 50 words (Table 3) found by probing WSJ88 (using the *Df* method) contains a fare number of important words like “market”, “share”, “trade”, “bank”, “stock”, “price”, etc., which are well suggestive of the overall subject of the database.

Table 3 gives the top 50 words in the learned dictionary when 100, 1000, and 3000 documents are examined, and the top 50 words in the real dictionary of WSJ88 database. It shows that as more documents are examined, the learned dictionary converges to be more close to the real dictionary. When 3000 documents have been seen (the third column), the two word sets of top 50 are nearly identical; only 4 out of 50 do not show in both lists. For the words that overlap, each word’s estimated rank is either the same as or very close to its true rank.

The curves of CACM and TREC-123 present similar patterns to that of WSJ88, and

Table 3: The top 50 words in the learned dictionary and real dictionary of WSJ88 database

learned (100 docs)	*learned (1000 docs)*	*learned (3000 docs)*	*real*
new	company	million	million
company	new	company	company
million	million	new	new
make	corp	share	say
corp	share	stock	share
base	stock	bank	market
two	base	market	trade
york	trade	say	stock
market	market	trade	bank
co	make	rate	price
share	co	billion	year
sale	two	price	corp
business	close	corp	billion
stock	exchange	month	make
time	month	sale	rate
high	york	make	president
plan	early	base	business
say	increase	increase	sale
trade	president	co	co
early	secure	fund	two
interest	report	president	month
month	hold	group	group
close	group	report	offer
product	sale	york	plan
america	billion	tax	report
call	america	manage	base
billion	bank	exchange	america
increase	price	two	high
manage	plan	business	time
operate	yesterday	offer	close
system	expect	early	york
buy	offer	plan	issue
group	unit	issue	govern
part	chairman	america	early
president	general	secure	work
price	operate	quarter	expect
expect	high	time	product
federal	business	close	manage
general	time	high	increase
issue	concern	unit	operate
long	executive	federal	unit
report	current	invest	quarter
term	issue	product	interest
work	end	interest	state
current	manage	hold	secure
large	service	operate	exchange
raise	interest	firm	official
result	total	service	bond
won	three	work	hold
yesterday	part	govern	federal

the above analysis and conclusions also apply to them. One noticeable difference between TREC-123 and the other two databases in terms of the goodness of the probing methods as shown in the curves is that, *Random* and *Atf* seem to be better than *Ctf* and *Df* for TREC-123, whereas the contrary is true for WSJ88 and CACM. This may be due to the large size and heterogeneous nature of TREC-123. Unlike the pattern using the other three evaluation metrics, *Random2* doesn't seem to be the best method in terms of mean squared *idf* error. This is because a term's *idf* is dependent on the number of documents in the collection, which can skew the result of mean squared *idf* error to some extent.

Figures 13, 14 and 15 compare the percentage of terms found, the *ctf* proportion, and the mean squared rank error, respectively, for the three test databases, all using the *Random2* method. The percentage of terms found is quite different among different databases, since their sizes vary widely. The pattern of *ctf* proportion for different databases indicates that the majority of frequent terms can be found when a certain number of documents are seen, no matter how big the database is. Around the point of 1000 documents, *ctf* proportion quickly grows to be higher than 0.88 for all three databases, and the curves level off thereafter, implying that, after a certain point, the terms found are mostly rare terms. Furthermore, rank errors for three databases converge to be very close to optimum (lower than 0.06), around the turning point of 1000 documents. This can be used to decide a proper probing termination condition (e.g., probing till 1000 documents are seen). This is a possible solution to the question of when to automatically stop probing.

5.3 Experiments with Microsoft Customer Support Database

To see how the probing idea works with real Web databases, a probing system was built to test the Web based Microsoft Customer Support Database. The system uses a Perl program to automatically send a query to the database, and to fetch a HTML document. It runs 10 queries per probe cycle, and gets the top 25 documents per query. The 10 initial probe words were chosen from the TREC-123 corpus, and INQUERY stopword list was used.

The probing system found 10,947 terms by examining 1,029 database documents using 600 probe queries. Figure 16 shows the number of terms found by probing versus the number of documents examined. Compared with the results of probing INQUERY databases, the pattern of dictionary growth is somewhat different, and *Atf* rather than *Random2* seems to be the best probing method (although not significantly better). This may be due to the Web HTML documents have rather different characteristics from the TREC collections with respect to document length, style, and nature of content.

The top 100 words (ranked by *ctf*) which are found by the *Random* method are given in Table 4 (along with their frequency statistics). They are shown in three lists ranked by their *ctf*, *df*, and *atf*, respectively. One can get a quick idea of what the database is about by looking at either of the lists, and conclude that it is about the popular Microsoft software systems. It seems that the list of words ranked by *atf* (the third list) has a better effect for browsing since the significant words, such as "excel", "foxpro", "microsoft", "nt", "access", "windows", etc., show up high in the top 50 list, and the list contains more content words.

There are some words which happen to appear in the top 50 but tell little about the content, such as "set", "please", etc. This is related to the stopword list used. They could have been removed by using a larger stopword list. However, there has to be a tradeoff in deciding which words should be counted as stopwords, since some unimportant words in one database may potentially be content words for others. For example, "word" is a non-significant term in general, but is used as a proper name here representing a perticular Microsoft software product, hence is an important term.

Word stems and their morphological variants are both included in the dictionary, since we didn't do stemming while parsing documents for efficiency reasons. It is possible to improve the results by using a stemming algorithm to reduce each word to its root form, but the overall speed of probing will be lowered.

Different database systems are based on different assumptions about tokenizing, case, stopwords, stemming, etc. This will effect the probing results and estimation of database content. For example, our probing system uniformly converts all terms to lower case, hence cannot distinguish terms that differ only in case, such as "Word" and "word", "Access" and "access", "Office" and "office", etc., in the Microsoft database.

Table 4: The top 50 words found by probing Microsoft Customer Support Database

ranked by *ctf*				ranked by *df*				ranked by *atf*			
term	*ctf*	*df*	*atf*	term	*ctf*	*df*	*atf*	term	*ctf*	*df*	*atf*
microsoft	5891	1027	5.736	microsoft	5891	1027	5.736	project	721	66	10.924
windows	2264	672	3.369	navigator	1046	1022	1.023	excel	525	60	8.750
information	2172	706	3.076	document	1573	1022	1.539	office	728	85	8.565
file	1679	345	4.867	reviewed	1105	1021	1.082	works	835	113	7.389
server	1585	218	7.271	corporation	1123	1021	1.100	server	1585	218	7.271
document	1573	1022	1.539	copy	1185	954	1.242	word	816	113	7.221
product	1420	365	3.890	reserved	978	949	1.031	table	478	72	6.639
beta	1376	276	4.986	rights	1030	948	1.086	printer	475	73	6.507
following	1189	351	3.387	april	971	933	1.041	foxpro	480	74	6.486
copy	1185	954	1.242	information	2172	706	3.076	database	679	111	6.117
files	1133	466	2.431	windows	2264	672	3.369	microsoft	5891	1027	5.736
corporation	1123	1021	1.100	support	986	478	2.063	object	451	80	5.637
reviewed	1105	1021	1.082	files	1133	466	2.431	user	731	138	5.297
error	1067	253	4.217	available	629	466	1.350	visual	986	187	5.273
navigator	1046	1022	1.023	help	681	445	1.530	beta	1376	276	4.986
rights	1030	948	1.086	article	998	445	2.243	service	588	118	4.983
article	998	445	2.243	change	728	413	1.763	basic	554	113	4.903
support	986	478	2.063	check	591	390	1.515	file	1679	345	4.867
visual	986	187	5.273	questions	812	383	2.120	nt	562	116	4.845
reserved	978	949	1.031	provided	501	381	1.315	field	506	107	4.729
april	971	933	1.041	please	483	381	1.268	access	847	186	4.554
version	919	292	3.147	product	1420	365	3.890	print	496	109	4.550
set	912	231	3.948	asked	601	361	1.665	data	752	174	4.322
release	895	302	2.964	frequently	598	360	1.661	internet	636	149	4.268
new	886	263	3.369	following	1189	351	3.387	error	1067	253	4.217
system	854	241	3.544	web	741	346	2.142	box	792	188	4.213
access	847	186	4.554	file	1679	345	4.867	articles	408	99	4.121
works	835	113	7.389	location	425	331	1.284	setup	565	138	4.094
word	816	113	7.221	page	811	328	2.473	mail	423	104	4.067
questions	812	383	2.120	release	895	302	2.964	users	384	95	4.042
page	811	328	2.473	version	919	292	3.147	set	912	231	3.948
box	792	188	4.213	note	519	282	1.840	application	584	149	3.919
data	752	174	4.322	beta	1376	276	4.986	product	1420	365	3.890
web	741	346	2.142	type	695	274	2.536	menu	695	181	3.840
user	731	138	5.297	computer	671	269	2.494	text	591	159	3.717
change	728	413	1.763	knowledge	673	268	2.511	software	478	132	3.621
office	728	85	8.565	new	886	263	3.369	code	387	107	3.617
project	721	66	10.924	error	1067	253	4.217	name	715	198	3.611
name	715	198	3.611	return	557	249	2.237	system	854	241	3.544
menu	695	181	3.840	system	854	241	3.544	dialog	471	134	3.515
type	695	274	2.536	set	912	231	3.948	command	445	127	3.504
select	682	222	3.072	open	629	226	2.783	following	1189	351	3.387
help	681	445	1.530	select	682	222	3.072	windows	2264	672	3.369
message	680	218	3.119	base	658	219	3.005	new	886	263	3.369
database	679	111	6.117	server	1585	218	7.271	settings	398	120	3.317
knowledge	673	268	2.511	problem	614	218	2.817	example	643	204	3.152
computer	671	269	2.494	message	680	218	3.119	version	919	292	3.147
base	658	219	3.005	want	551	215	2.563	message	680	218	3.119
example	643	204	3.152	these	474	211	2.246	information	2172	706	3.076
internet	636	149	4.268	updated	501	210	2.386	select	682	222	3.072

6 Conclusion

This paper proposes the new idea of probing a database via a search interface in order to find out the contents of the hidden collection. The probing approach can automatically build an accurate model that represents the content of the database, for use in colletion selection in distributed retrieval. The words found by probing can give us an idea of what's in the database, which is also very useful information. The effectiveness of probing is demonstrated in experiments with the INQUERY information retrieval system and three TREC collections, as well as a real Web based database service.

The experimental results are encouraging because a simple probing approach was quite effective in accurately estimating collection content. Overall effectiveness of the probing methods studied is good, and random samples seem to be better. The probing techniques are effective with databases that vary widely in size and nature. Collection models can be built without cooperation, which is an important contribution.

The database size, or the number of documents in the collection (N), is unknown but is an important parameter. The possibility of estimating N through probing is an open problem. By definition, a term's unscaled *idf* is still related to N, hence is collection-specific. Since the *idf* is not entirely independent of database size, the results of mean squared *idf* error were skewed by database size to some extent.

Only one real Web database was used to test the probing techniques. As can be seen from the results based on the Microsoft database, real Web databases present a somewhat different probing pattern than INQUERY databases. The Web environment is highly heterogenous in that databases differ significantly from each other in size, document length and content. Further experiments with more Web databases are needed to see how the probing system performs in the dynamic networked environments, and to draw a general conclusion on which probing method is the best.

The evaluation metrics we used are only applicable to research databases whose original data are readily available. It's unclear how to evaluate the effectiveness of probing real Web databases. Without knowing the actual database content and statistics as a basis for comparison, we only examined the number of words found by probing, which seems to be a coarse evaluation. More accurate evaluation measures suitable to real Web environment are desirable.

There are other variations in probing method which we haven't tested. For example, one could use phrase queries instead of one-word queries; use less frequent words instead of top words as probes; or choose probe words from the set of new words found in each probe cycle, instead of from the whole learned dictionary. Although we believe that varying the experimental parameters won't dramatically change the effectiveness of probing, more work is needed to see if further improvements are possible.

Acknowledgements

This material is based on work supported in part by the National Science Foundation, Library of Congress and Department of Commerce under cooperative agreement number EEC-9209623,and also supported in part by United States Patent and Trademark Office and Defense Advanced Research Projects Agency/ITO under ARPA order number D468, issued by ESC/AXS contract number F19628-95-C-0235. Any opinions, findings and conclusions or recommendations expressed in this material are the author(s) and do not necessarily reflect those of the sponsor(s).

References

[1] J. P. Callan, W. B. Croft, and S. M. Harding. The INQUERY retrieval system. In *Proceedings of the Third International Conference on Database and Expert Systems Applications*, pages 78–83, Valencia, Spain, 1992. Springer-Verlag.

[2] J. P. Callan, Z. Lu, and W. B. Croft. Searching distributed collections with inference networks. In *Proceedings of 18th ACM SIGIR International Conference on Research and Development in Information Retrieval*, pages 21–29, Seattle, WA, 1995.

[3] J. C. French, J. C. Powell, C. L. Viles, T. Emmitt, and K. J. Prey. Evaluating database selection techniques: A testbed and experiment. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Melbourne, Australia, 1998.

[4] L. Gravano, Kevin Chang, H. García-Molina, and Andreas Paepcke. STARTS Stanford protocal proposal for Internet retrieval and search. Technical report, Computer Science Department, Stanford University, 1996.

[5] L. Gravano and García-Molina. Generalizing GLOSS to vector-space databases and broker hierarchies. In *Proceedings of the 21st International Conference on Very Large Databases (VLDB)*, Zurich, Switzerland, 1995.

[6] L. Gravano, H. García-Molina, and A. Tomasic. The effectiveness of GLOSS for the text database discovery problem. In *Proceedings of SIGMOD 94*, pages 126–137. ACM, May 1994.

[7] L. Gravano, H. García-Molina, and A. Tomasic. Precision and recall of GLOSS estimators for database discovery. Technical Report STAN-CS-TN-94-10, Stanford University, 1994.

[8] H. P. Luhn. The automatic creation of literature abstracts. *IBM Journal of Research and Development*, 2:159–165, 1958.

[9] H. R. Turtle and W. B. Croft. Efficient probabilistic inference for text retrieval. In *RIAO 3 Conference Proceedings*, pages 644–661, Barcelona, Spain, April 1991.

[10] H. R. Turtle and W. B. Croft. Evaluation of an inference network-based retrieval model. *ACM transactions on Information Systems*, 9(3):187–222, 1991.

[11] Jinxi Xu and J. P. Callan. Effective retrieval with distributed collections. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Melbourne, Australia, 1998.

[12] H. P. Zipf. *Human Behaviour and the Principle of Least Effort.* Addison-Wesley, Cambridge, Massachusetts, 1949.

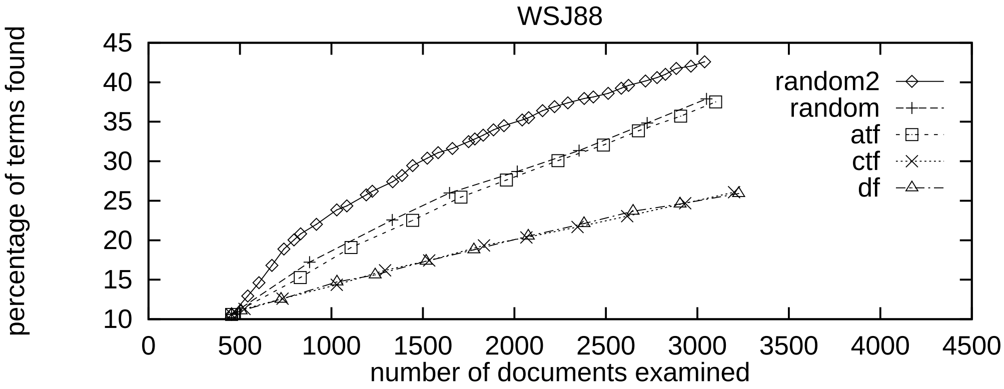

Figure 1: WSJ88: percentage of terms found by probing

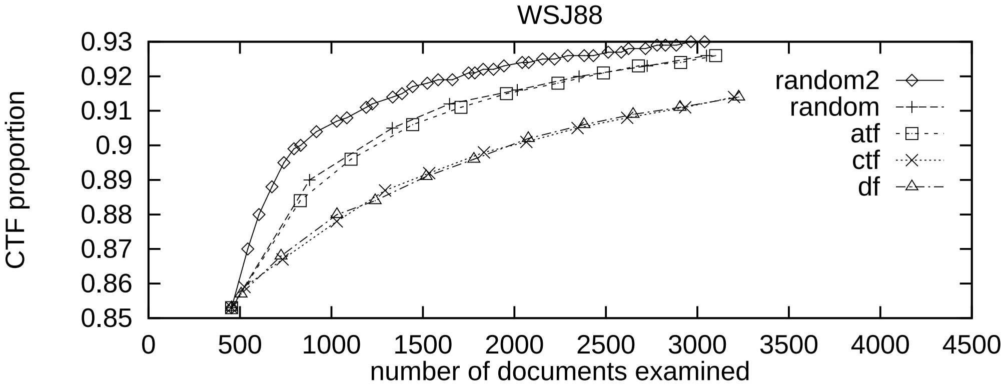

Figure 2: WSJ88: CTF proportion

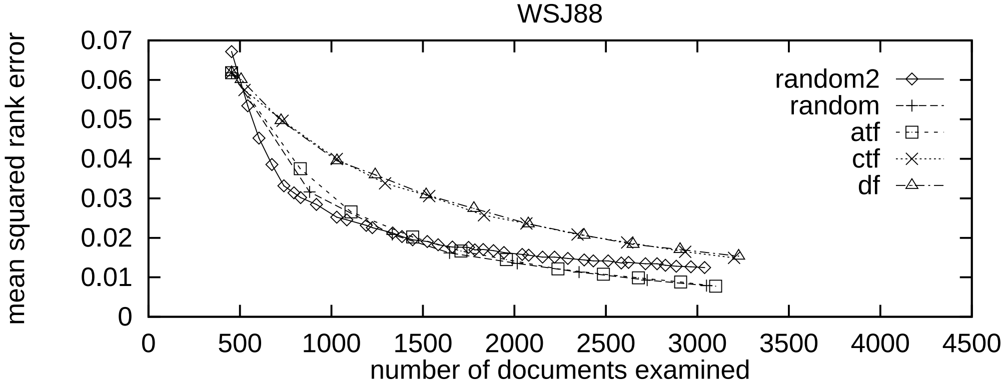

Figure 3: WSJ88: mean squared rank error

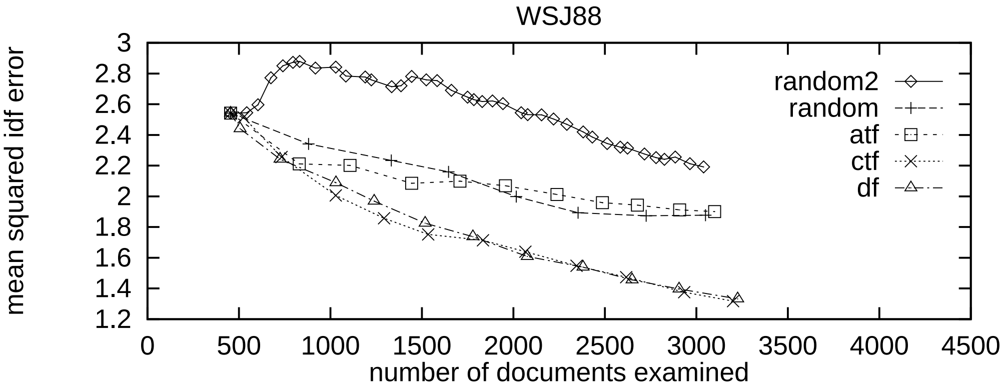

Figure 4: WSJ88: mean squared idf error

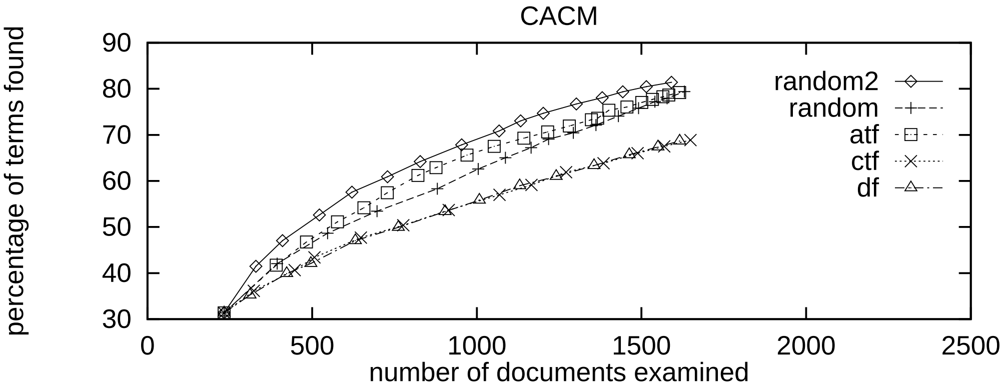

Figure 5: CACM: percentage of terms found by probing

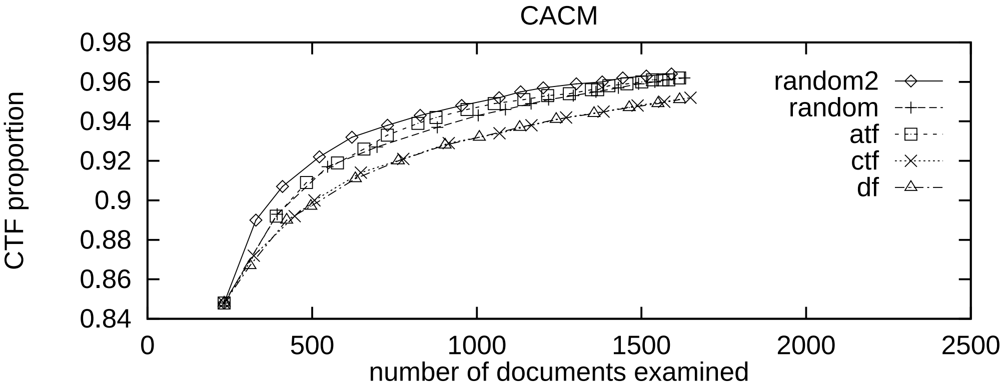

Figure 6: CACM: CTF proportion

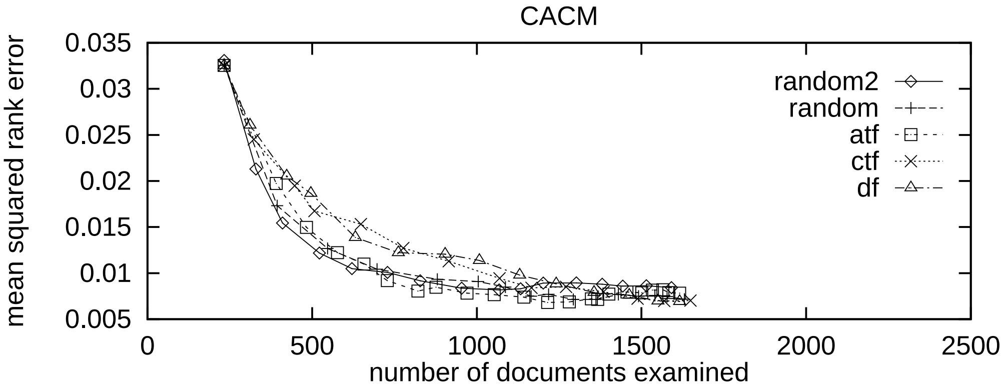

Figure 7: CACM: mean squared rank error

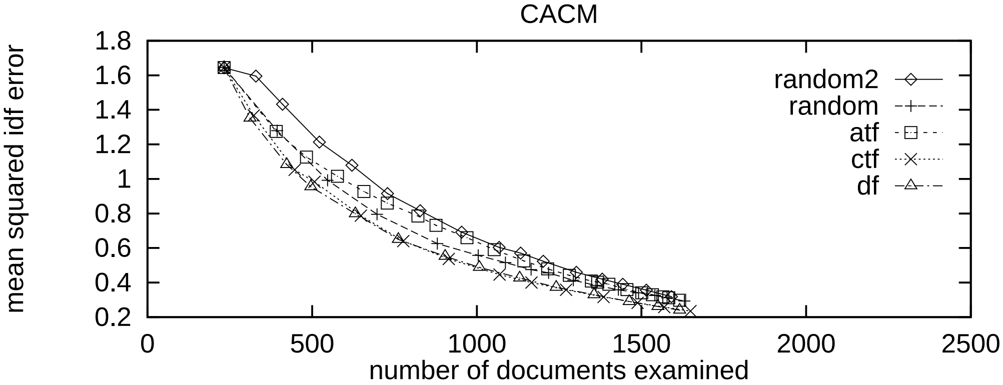

Figure 8: CACM: mean squared idf error

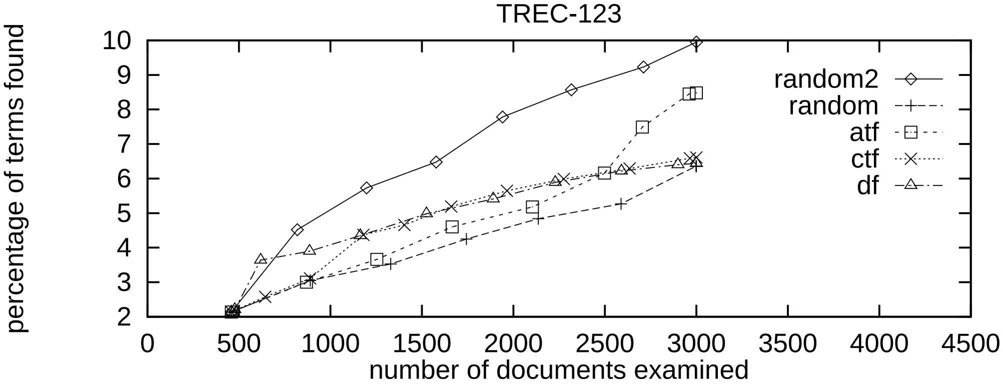

Figure 9: TREC-123: percentage of terms found by probing

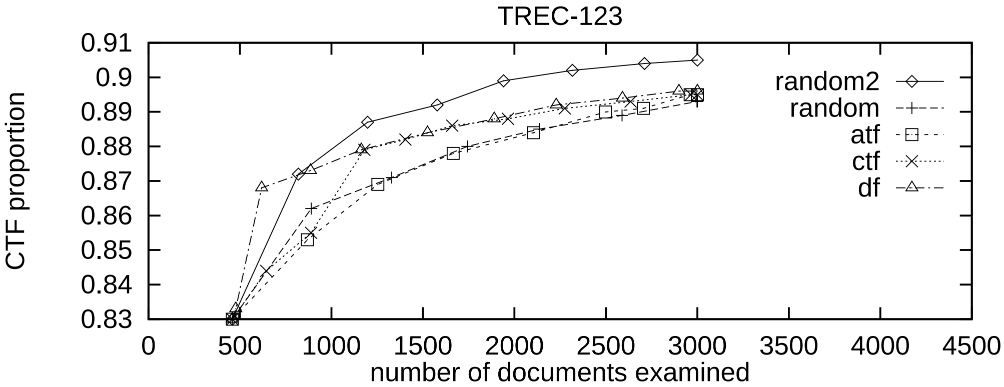

Figure 10: TREC-123: CTF proportion

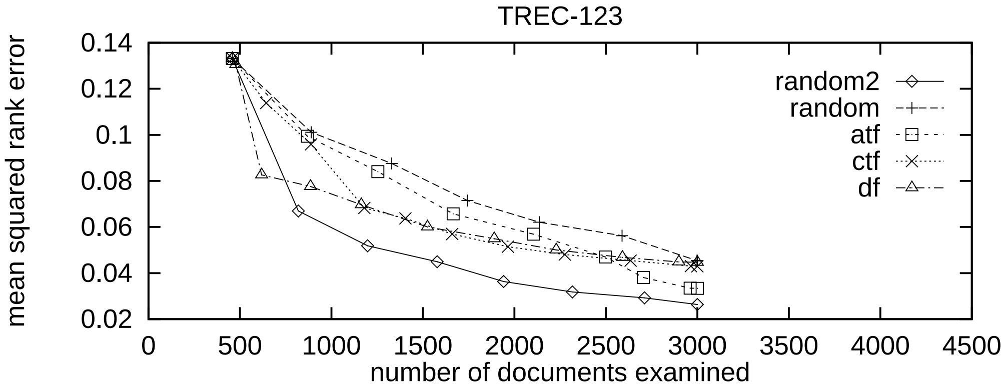

Figure 11: TREC-123: mean squared rank error

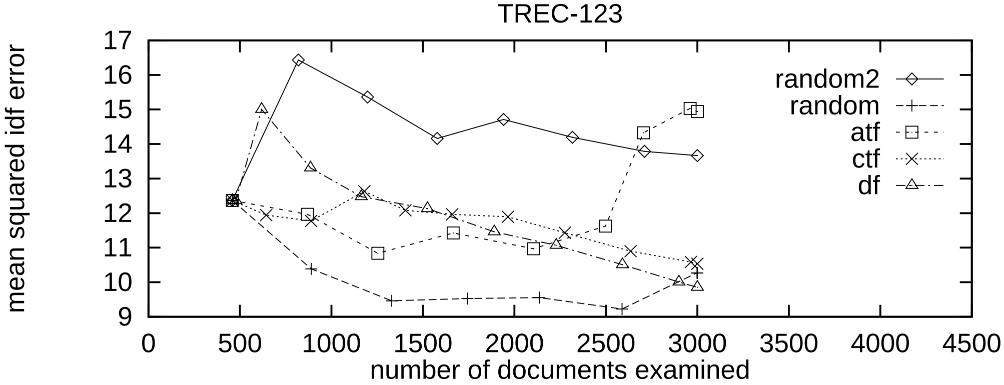

Figure 12: TREC-123: mean squared idf error

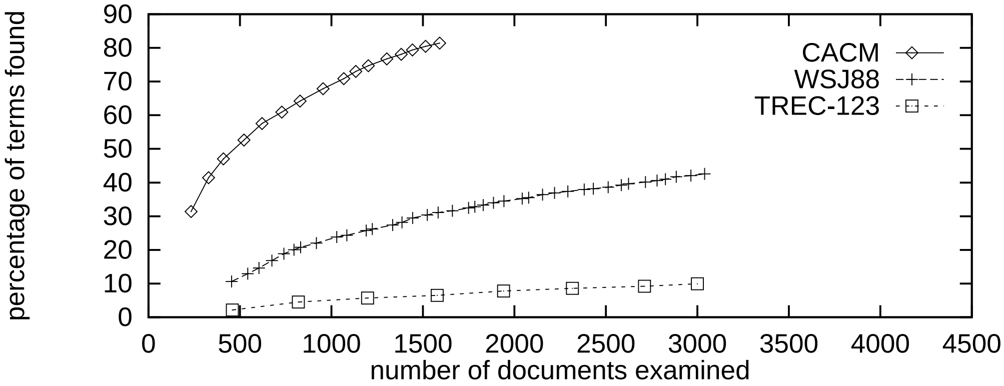

Figure 13: % of terms found for CACM, WSJ88, and TREC-123 (*Random2* method)

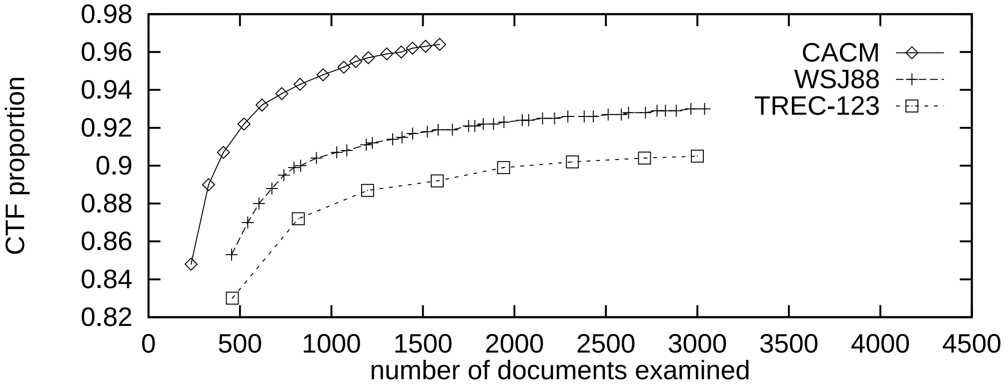

Figure 14: CTF proportion for CACM, WSJ88, and TREC-123 (*Random2* method)

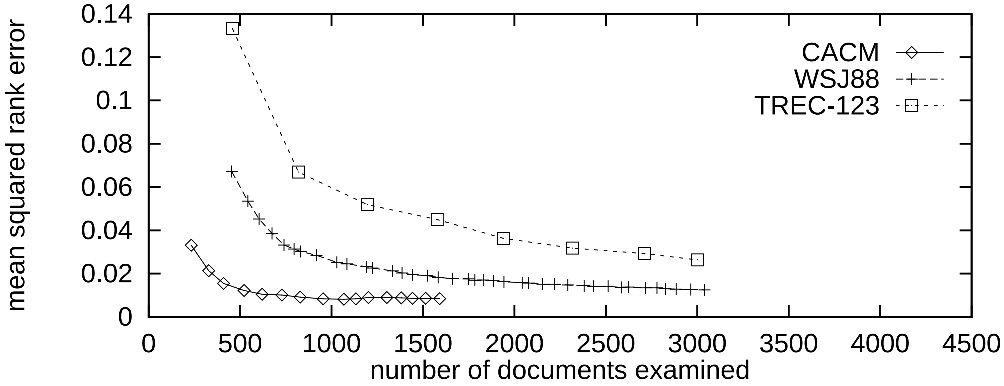

Figure 15: rank error for CACM, WSJ88, and TREC-123 (*Random2* method)

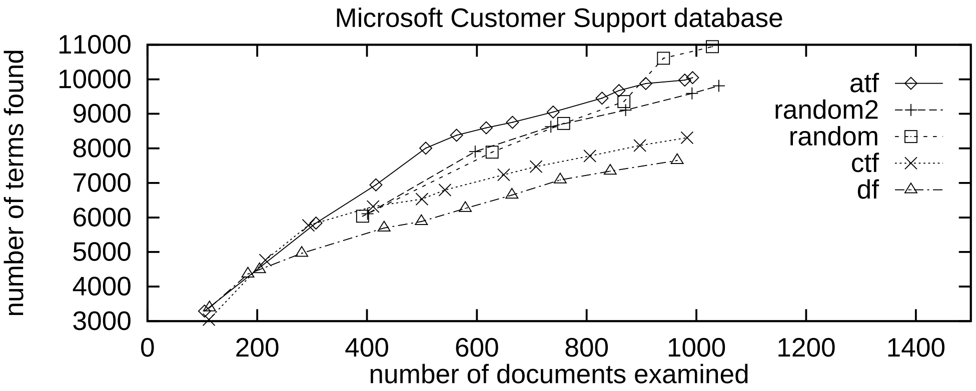

Figure 16: Microsoft Customer Support Database